

Efficient and Provable Robustness Verification of Neural Networks

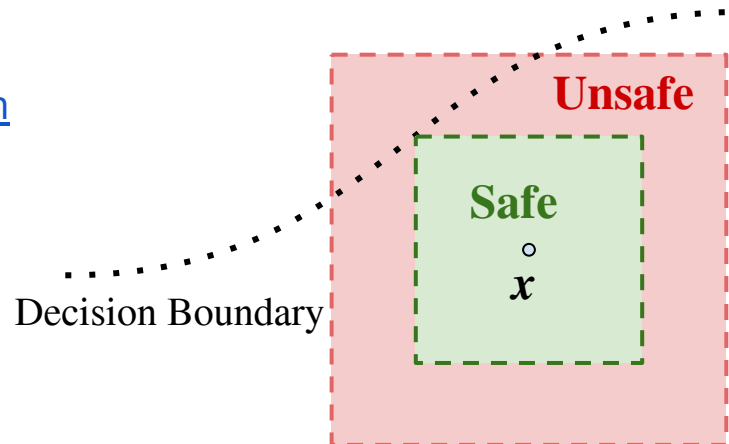
Huan Zhang

CMU

06/16/2021

huan@huan-zhang.com

VALSE 视觉与学习青年学者研讨会
ONLINE Vision And Learning SEminar



Adversarial Examples and Robustness

- Neural networks are vulnerable to small adversarial perturbations (“adversarial examples”)

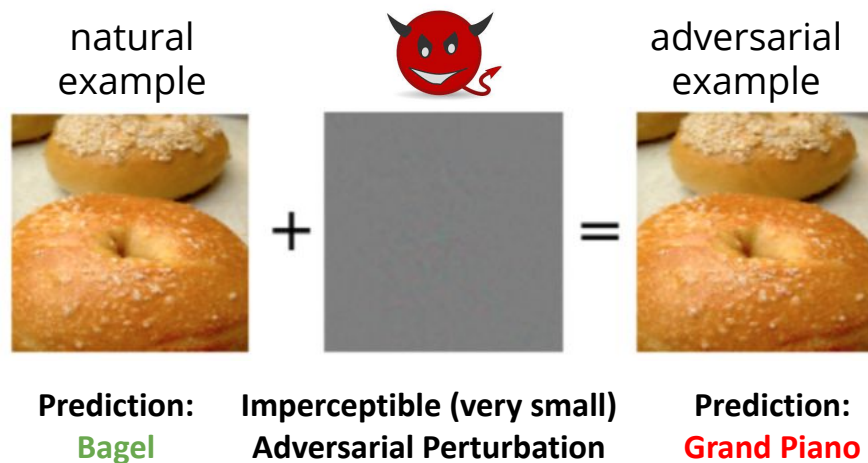
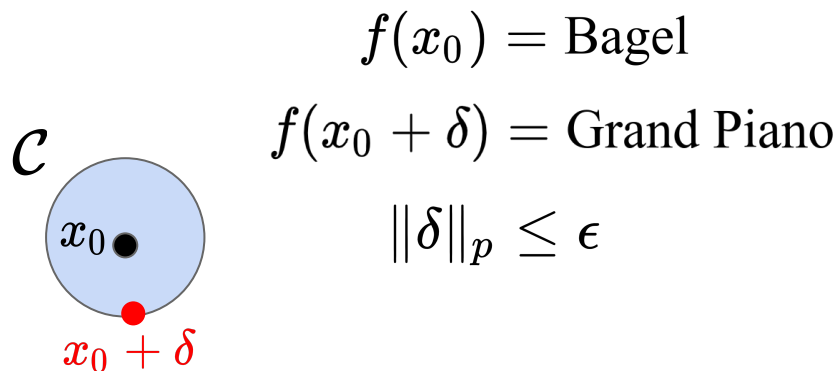


Image from “ZOO: Zeroth Order Optimization based Black-box Attacks to Deep Neural Networks”, Pin-Yu Chen*, Huan Zhang*, Yash Sharma, Jinfeng Yi, Cho-Jui Hsieh. (*Equal contribution)

Adversarial Examples and Robustness

- Since the discovery of adversarial examples, many defenses have been proposed
- Many adversarial defenses are broken under stronger and adaptive attacks (e.g., Athalye et al., 2018)

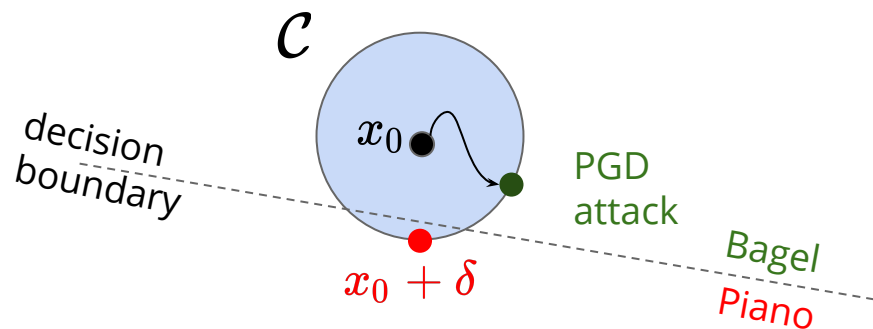


**Obfuscated Gradients Give a False Sense of Security:
Circumventing Defenses to Adversarial Examples**

Anish Athalye^{*1} Nicholas Carlini^{*2} David Wagner²

Motivations for Robustness Verification

- Can we formally *prove* that a classifier is robust? (e.g., guarantee that no adversarial example exists)
- Just using attacks is not not sufficient; it cannot always guarantee to find an adversarial example even it exists



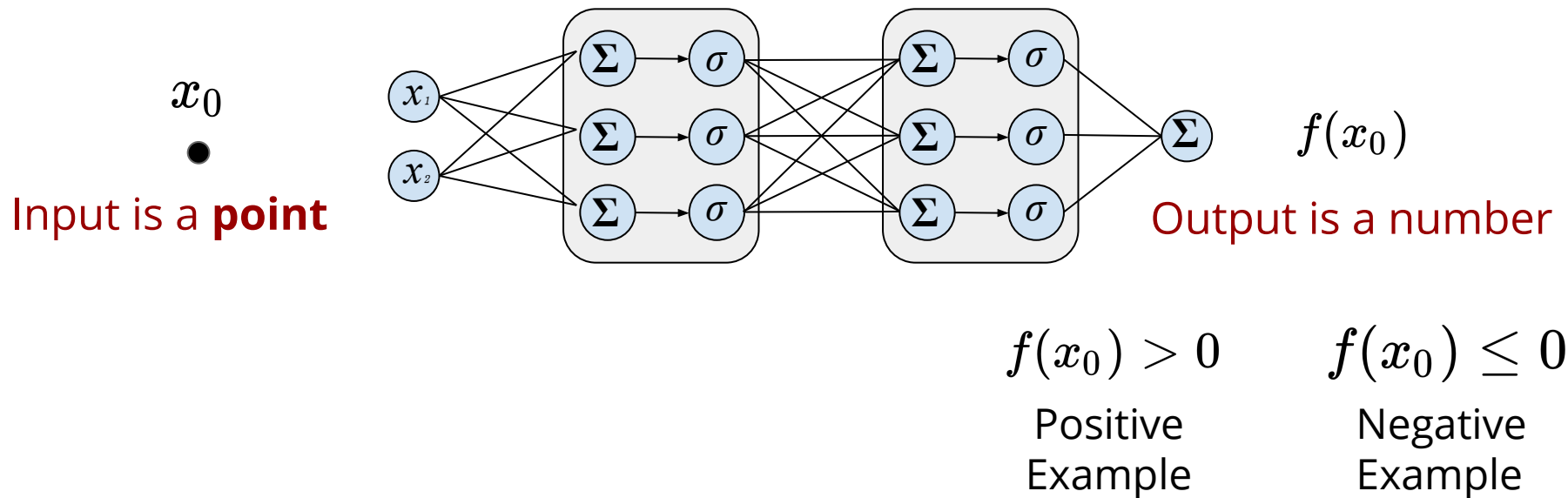
Prediction:
Bagel

+ Any Bounded
Adversarial Perturbation

= Prediction:
Bagel

Give *Provable* Robustness Guarantees

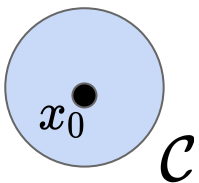
Consider binary classification with a simple NN



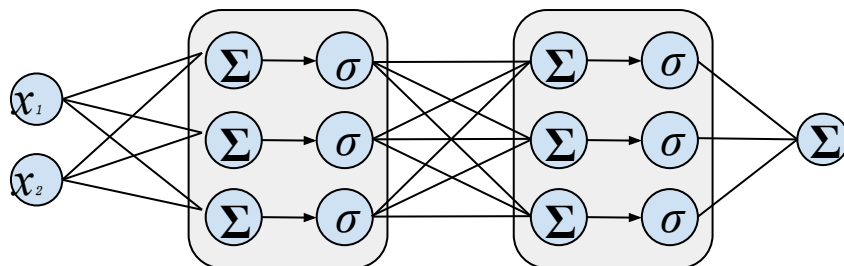
Give *Provable* Robustness Guarantees

Suppose $f(x_0) > 0$. Can we verify this property:

$$f(x) > 0, \forall x \in \mathcal{C}$$



Input is a set



$$f_L^* \leq f(x) \leq f_U^*$$

Output is a range

Must consider a set of infinite points as the input of the NN.

The Robustness Verification (鲁棒性验证) Problem

Assuming $f(x_0) > 0$, we solve the non-convex problem:

$$f^* = \min_{x \in \mathcal{C}} f(x)$$

\mathcal{C} is usually a set “around” x_0 , e.g., $\mathcal{C} := \{x \mid \|x - x_0\|_p \leq \epsilon\}$

- If $f^* < 0$ then we can flip the label (not robust!)
- If $f^* > 0$ then the classifier is verifiably robust in \mathcal{C}

The Robustness Verification (鲁棒性验证) Problem

Assuming $f(x_0) > 0$, we solve the non-convex problem:

$$f^* = \min_{x \in \mathcal{C}} f(x)$$

\mathcal{C} is usually a set “around” x_0 , e.g., $\mathcal{C} := \{x \mid \|x - x_0\|_p \leq \epsilon\}$

- **Complete** verification (完备验证): solve exactly. E.g., using mixed integer programming (**NP-complete**), hours even on small NNs
- **Incomplete** (or relaxed) verification (不完备验证): find a *lower bound* of f^* . If **lower bound** > 0 , the network is verifiably robust.

Robustness Verification vs. Adversarial Attacks

- Attack and verification algorithms evaluate robustness from two distinct perspectives: upper bound (attack) and lower bound (verification)

Incomplete verification algorithm

stronger →

Complete verification algorithm

← stronger

Attack algorithm

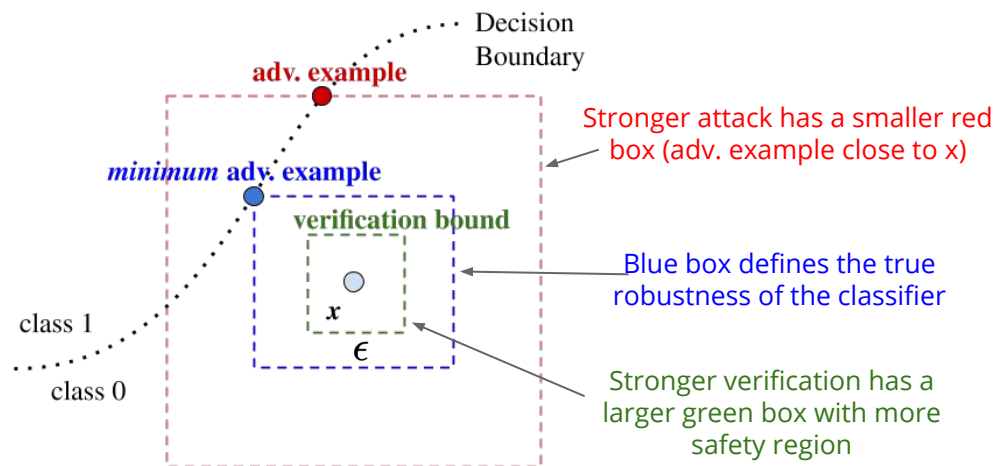
(wants the green box to be as large as possible)

(very slow, but guarantees to find the minimum adv. example)

(wants the red box to be as small as possible)

$$f^* = \min_{x \in \mathcal{C}} f(x)$$

A binary search on the perturbation radius ϵ can be conducted to find the largest verifiable region (green box)

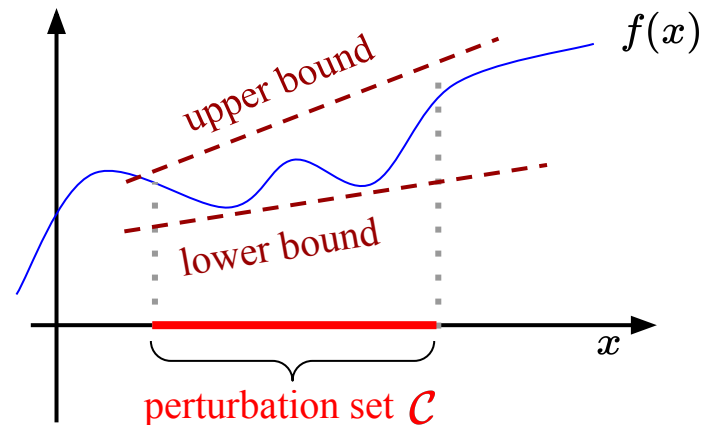


CROWN: An Incomplete Verification Algorithm

- We want to find a lower bound for this problem efficiently:

$$f_L \leq f^* = \min_{x \in \mathcal{C}} f(x)$$

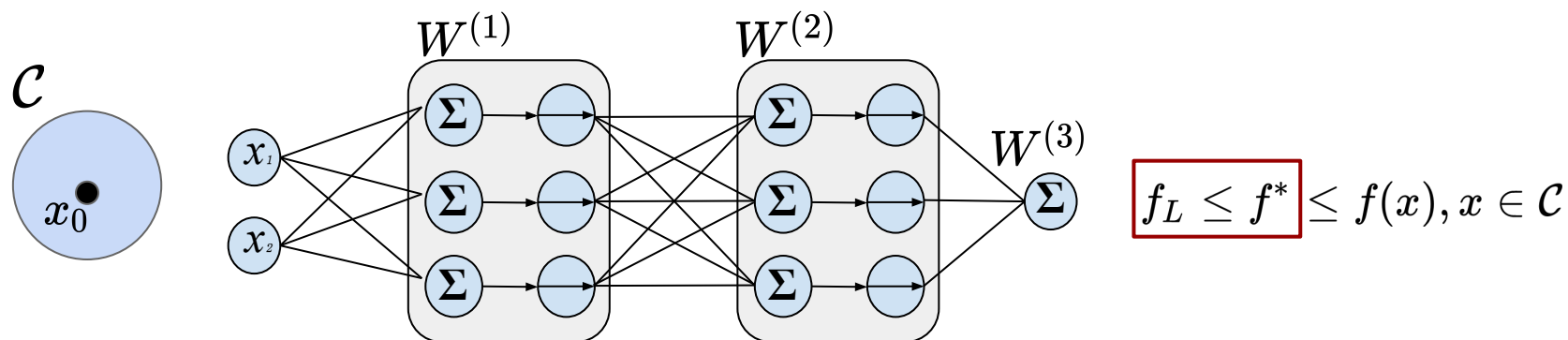
- $f_L > 0 \Rightarrow f^* > 0$, so no adversarial example exists if $f_L > 0$
- **CROWN** (Zhang et al. 2018) is an efficient, bound propagation (限界传播) based algorithm to find f_L



Efficient Neural Network Robustness Certification with General Activation Functions, **Huan Zhang***, Tsui-Wei Weng*, Pin-Yu Chen, Cho-Jui Hsieh, Luca Daniel. (* Equal contribution). *NIPS 2018*.

<https://arxiv.org/pdf/1811.00866.pdf>

Find the Lower Bound on Feed-forward Networks

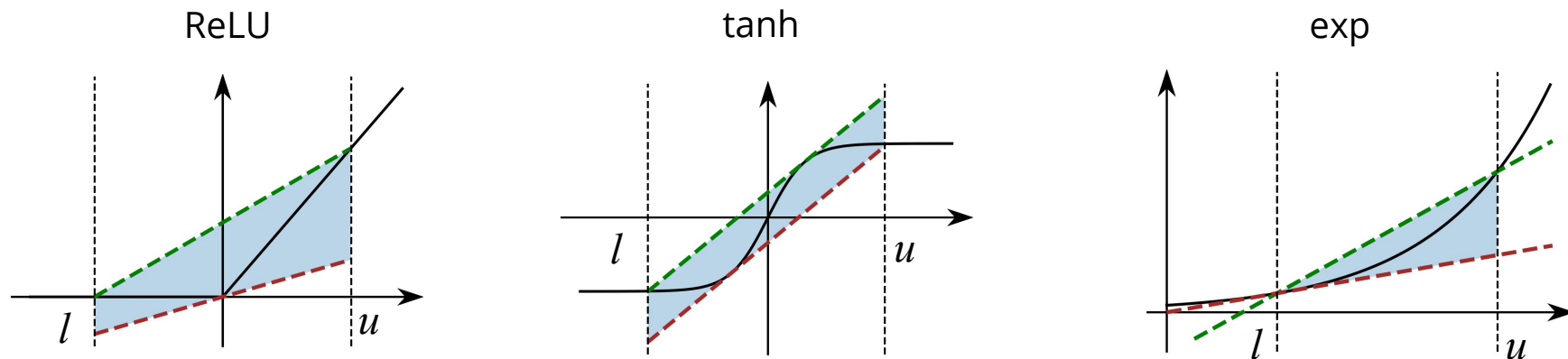


- If there are no non-linear operations (e.g., ReLUs), all weights can be multiplied together

$$f(x) = W^{(3)} W^{(2)} W^{(1)} x = a^\top x$$

- Bounds for linear functions are easy (e.g., Cauchy-Schwarz or Hölder's inequality)

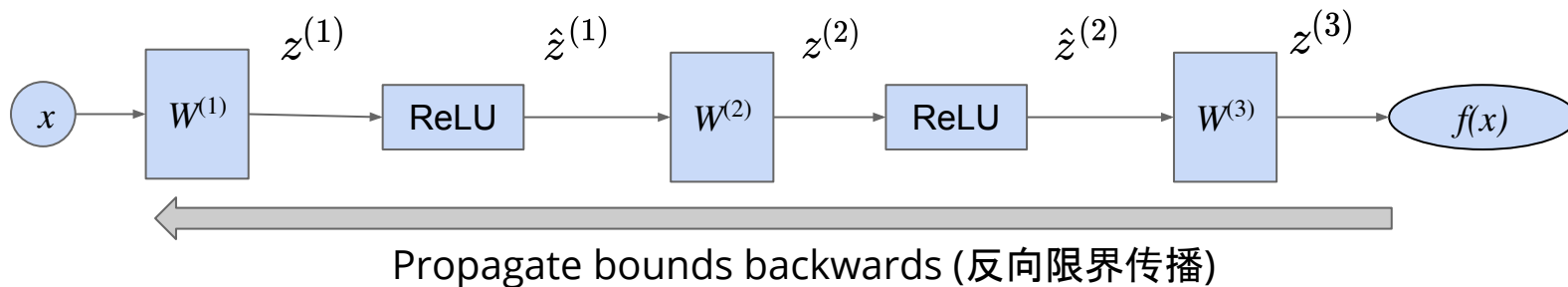
Linear Relaxations of Non-linear Functions



- Idea: use two linear **upper** and **lower** bounds to replace non-linear units, to obtain linear bounds for the entire network
- Relaxations depend on input bounds (pre-activation bounds) for the non-linear function, denoted as l and u

CROWN Backward Bound Propagation

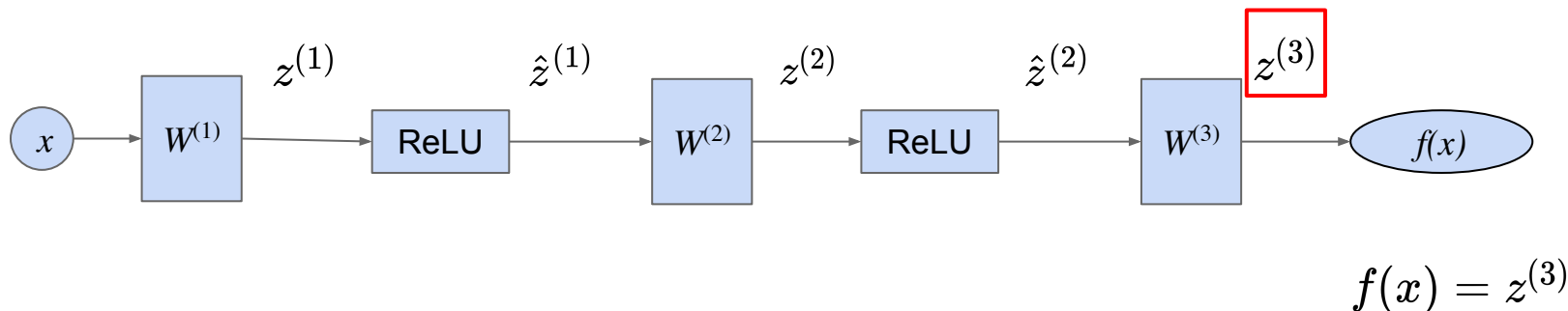
- In CROWN, we **propagate a linear lower bound** for **output neuron w.r.t. hidden or input neuron**.



- Goal: get a lower bound for $f(x) > 0, \forall x \in \mathcal{C}$

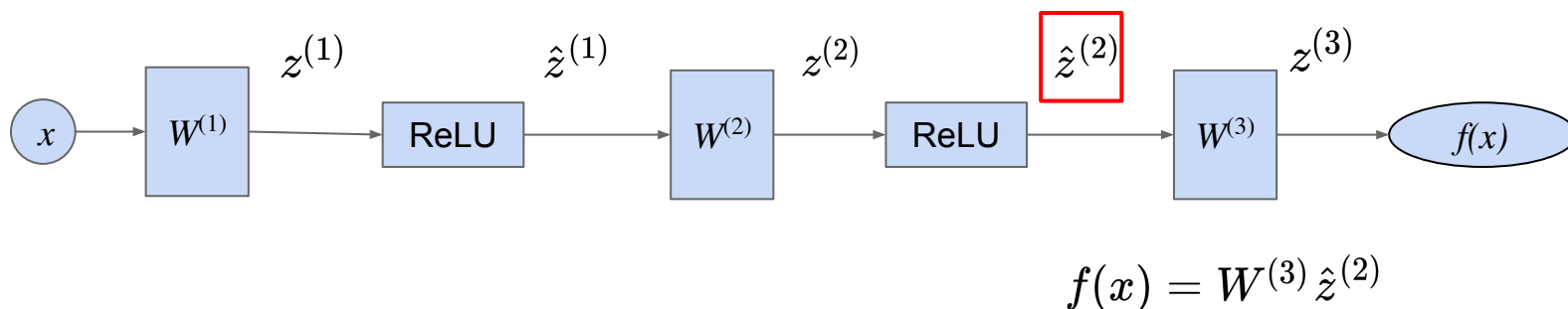
CROWN Backward Bound Propagation

- In CROWN, we **propagate a linear lower bound** for **output neuron w.r.t. hidden or input neuron**.



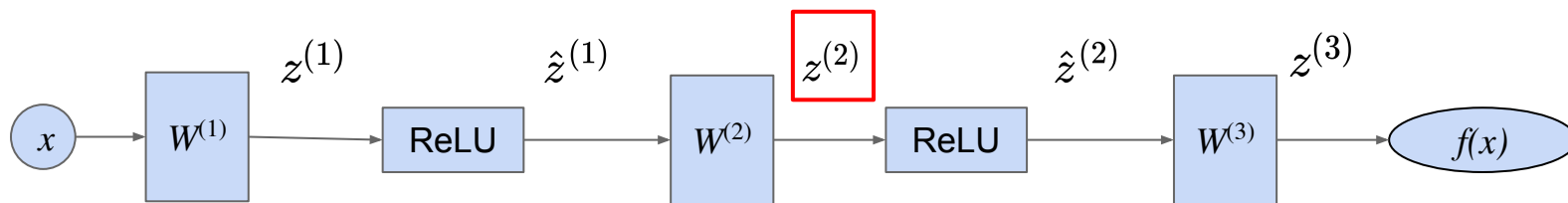
CROWN Backward Bound Propagation

- In CROWN, we **propagate a linear lower bound** for **output neuron w.r.t. hidden or input neuron**.



CROWN Backward Bound Propagation

- In CROWN, we **propagate a linear lower bound** for **output neuron w.r.t. hidden or input neuron**.



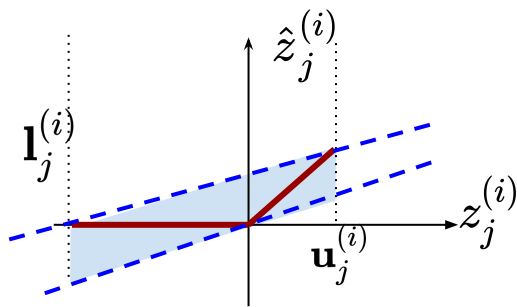
$$f(x) \geq W^{(3)} D^{(2)} z^{(2)} + \text{const.}$$

Encountered a nonlinear operation.

A diagonal matrix D reflects the relaxation of ReLU neurons will be used.

CROWN bound propagation

- How to design D so the lower and upper bounds are maintained?
- First step: linear lower and upper bound the non-linear function



$$\underline{a}_j^{(i)} z_j^{(i)} + \underline{b}_j^{(i)} \leq \hat{z}_j^{(i)} \leq \bar{a}_j^{(i)} z_j^{(i)} + \bar{b}_j^{(i)}$$

For ReLU:

$$\hat{z}_j^{(i)} \leq \frac{\mathbf{u}_j^{(i)}}{\mathbf{u}_j^{(i)} - \mathbf{l}_j^{(i)}} \left(z_j^{(i)} - \mathbf{l}_j^{(i)} \right)$$

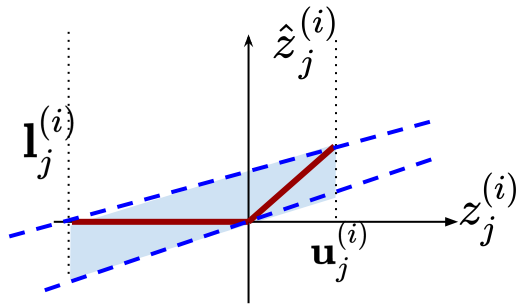
$$\hat{z}_j^{(i)} \geq \alpha_j^{(i)} z_j^{(i)} \quad (0 \leq \alpha_j^{(i)} \leq 1)$$

$\mathbf{l}_j^{(i)} \leq z_j^{(i)} \leq \mathbf{u}_j^{(i)}$ are pre-activation bounds

Adjustable!

CROWN bound propagation

- How to design D so the lower and upper bounds are maintained?
- Second step: choose the lower or upper bound based the sign of the matrix before D , which is $W^{(3)}$ in this case



We want the **lower bound**:

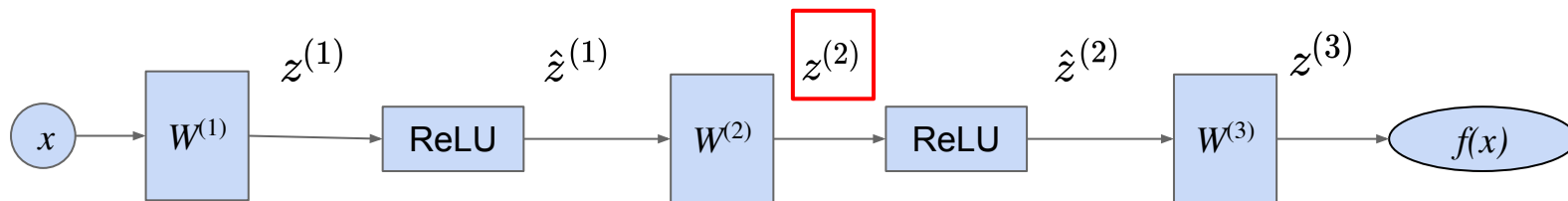
For numbers in $A \geq 0$, take upper bound of ReLU;
For numbers in $A < 0$, take lower bound of ReLU.

$$\underline{D}_{j,j}^{(2)} = \begin{cases} \underline{a}_j^{(2)}, & W_{1,j}^{(3)} \geq 0 \\ \bar{a}_j^{(2)}, & W_{1,j}^{(3)} < 0 \end{cases}$$

$$\underline{a}_j^{(i)} z_j^{(i)} + \underline{b}_j^{(i)} \leq \hat{z}_j^{(i)} := \text{ReLU}(z_j^{(i)}) \leq \bar{a}_j^{(i)} z_j^{(i)} + \bar{b}_j^{(i)}$$

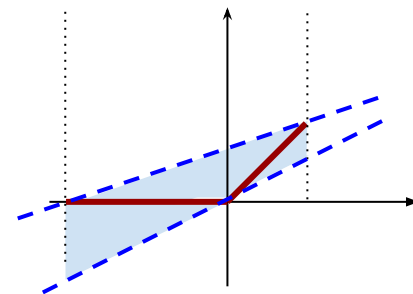
CROWN Backward Bound Propagation

- In CROWN, we **propagate a linear lower bound** for **output neuron w.r.t. hidden or input neuron**.



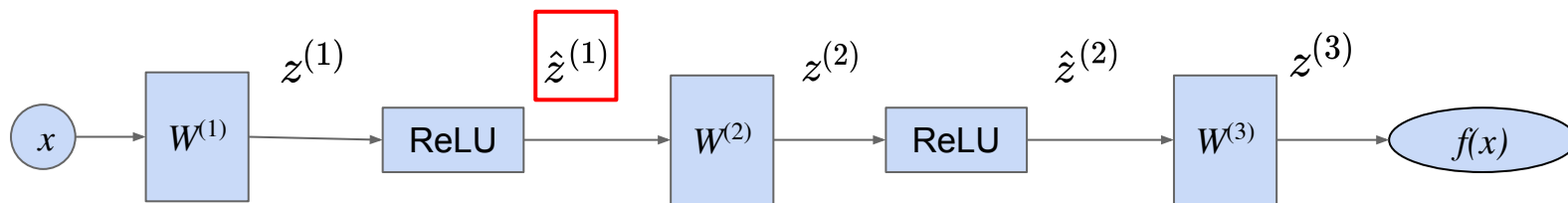
$$f(x) \geq W^{(3)} D^{(2)} z^{(2)} + \text{const.}$$

$D^{(2)}$ depends on the signs in $W^{(3)}$, and the linear relaxation of ReLU neuron to make the inequality hold



CROWN Backward Bound Propagation

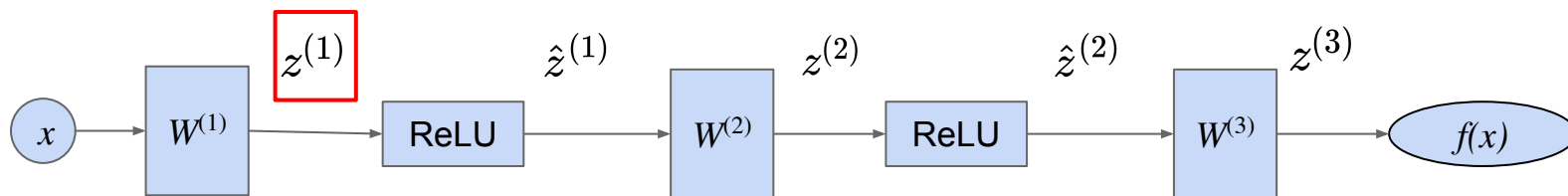
- In CROWN, we **propagate a linear lower bound** for **output neuron w.r.t. hidden or input neuron**.



$$f(x) \geq W^{(3)} D^{(2)} W^{(2)} \hat{z}^{(1)} + \text{const.}$$

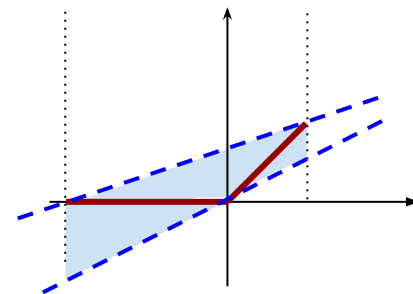
CROWN Backward Bound Propagation

- In CROWN, we **propagate a linear lower bound** for **output neuron w.r.t. hidden or input neuron**.



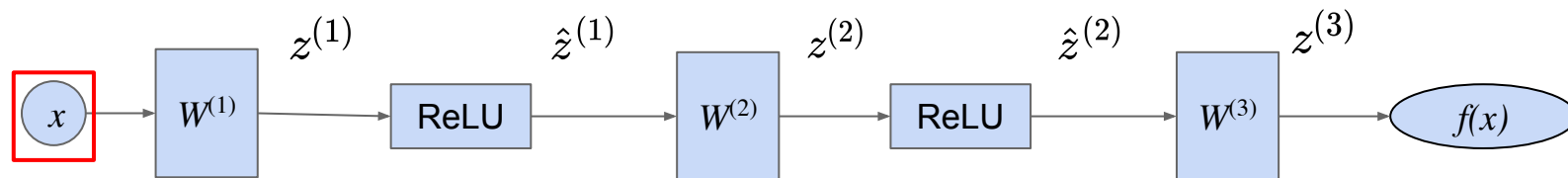
$$f(x) \geq W^{(3)} D^{(2)} W^{(2)} D^{(1)} z^{(1)} + \text{const.}$$

$D^{(1)}$ depends on the signs in $W^{(3)} D^{(2)} W^{(2)}$, and the linear relaxation of ReLU neuron to make the inequality hold



CROWN Backward Bound Propagation

- In CROWN, we **propagate a linear lower bound** for **output neuron w.r.t. hidden or input neuron**.

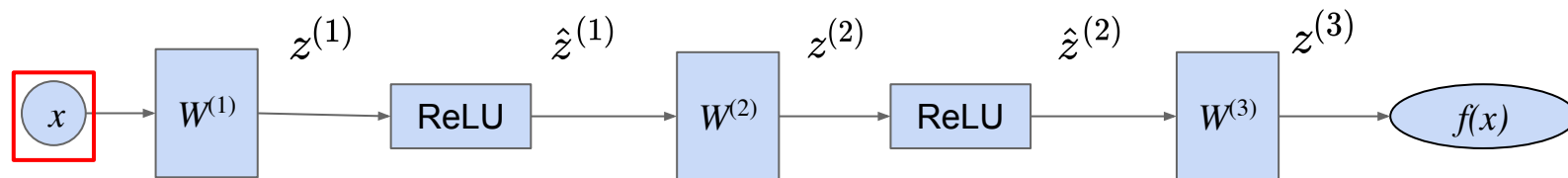


$$f(x) \geq W^{(3)} D^{(2)} W^{(2)} D^{(1)} W^{(1)} x + \text{const.}$$

← Propagate bounds backwards

CROWN Backward Bound Propagation

- In CROWN, we **propagate a linear lower bound** for **output neuron w.r.t. hidden or input neuron**.



$$f(x) \geq W^{(3)} D^{(2)} W^{(2)} D^{(1)} W^{(1)} x + \text{const.}$$

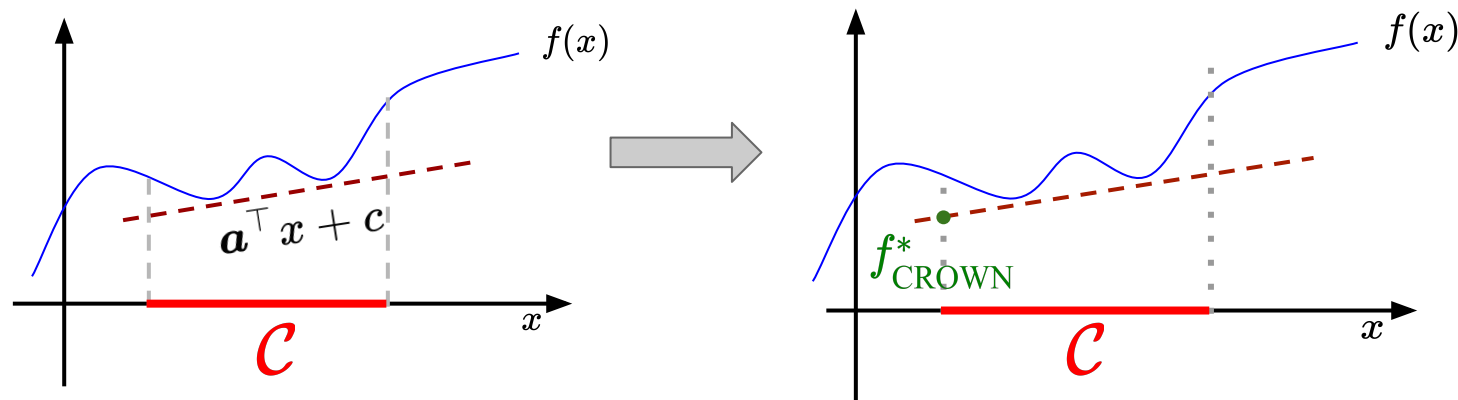
CROWN bound: $\min_{x \in \mathcal{C}} f(x) \geq \min_{x \in \mathcal{C}} \mathbf{a}_{\text{CROWN}}^\top x + \mathbf{c}_{\text{CROWN}} := \min_{x \in \mathcal{C}} f_{\text{CROWN}}(x)$

Where $\mathbf{a}_{\text{CROWN}}$ and $\mathbf{c}_{\text{CROWN}}$ can be **computed efficiently** on GPUs in a backward manner, as a function of NN weights,

The CROWN Lower Bound

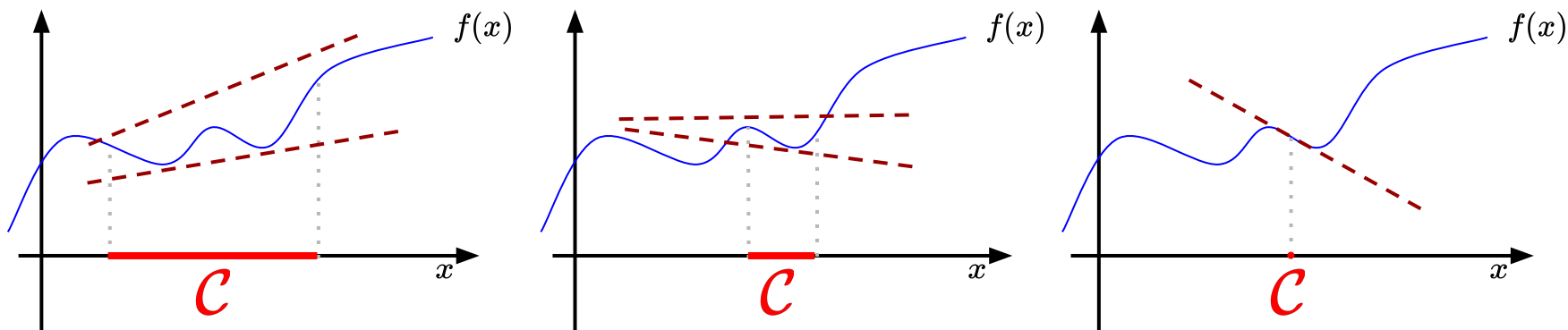
Final lower bound can be computed based on the CROWN linear bound by solving a easier linear optimization problem (which can be solved in closed form for ℓ_p norm perturbations):

$$f_{\text{CROWN}}^* = \min_{x \in \mathcal{C}} \mathbf{a}_{\text{CROWN}}^\top \mathbf{x} + c_{\text{CROWN}}$$



CROWN vs. Gradient from Backpropagation

- Time complexity similar to backpropagation (assuming known pre-activation bounds); both operate backwards on the graph.
- CROWN can be seen as a “generalized” gradient; when there is no perturbation, bounds become the gradient



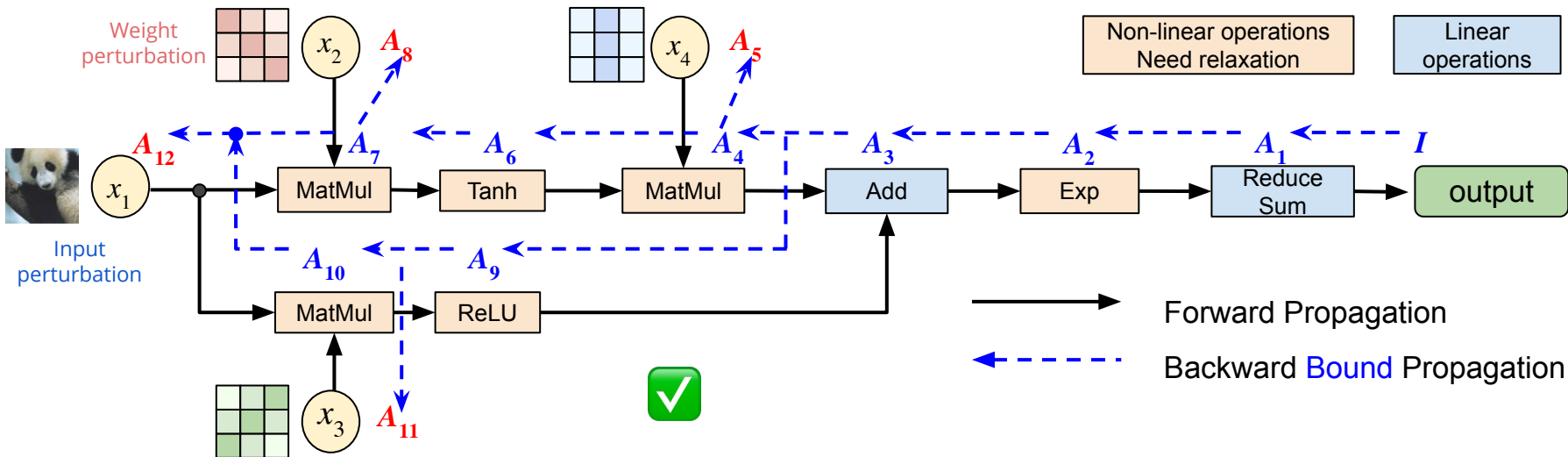
Challenges of CROWN

- Hard to understand (not friendly for people outside of this field...)
- Hard to derive (Feedforward NN is okay, but how about LSTM, Transformer, ResNet, DenseNet, graph neural networks.....)
- Hard to implement (for a new type of network, needs to re-derive and re-implement)

Robustness Verification for Transformers. Zhouxing Shi, **Huan Zhang**, Kai-Wei Chang, Minlie Huang, Cho-Jui Hsieh. ICLR 2020. <https://openreview.net/pdf?id=BJxwPJHFwS>

Generalization of CROWN

- We generalize CROWN to a **graph algorithm** to run on **general** computational graphs (a superset of many existing algorithms on verifying the robustness of LSTMs, Transformers, etc)



Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. Kaidi Xu*, Zhouxing Shi*, Huan Zhang* (*Equal contribution), Yihan Wang, Minlie Huang, Kai-Wei Chang, Bhavya Kailkhura, Xue Lin, Cho-Jui Hsieh. NeurIPS 2020.

The Propagation Can Be Hidden To Users...

```
from auto_LiRPA import BoundedModule, BoundedTensor, PerturbationLpNorm
```

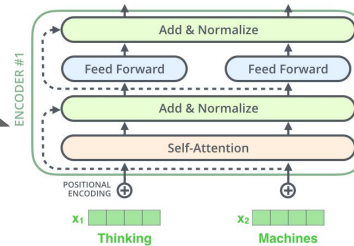
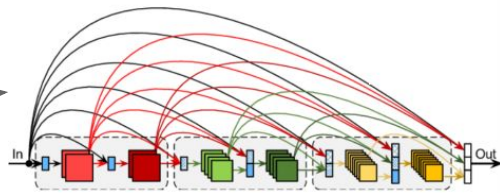
```
# Define computation as a nn.Module
class MyModel(nn.Module):
    def forward(self, x):
        # Define your computation here
```

```
model = MyModel()
my_input = load_a_batch_of_data()
# Wrap the model with auto_LiRPA
model = BoundedModule(model, my_input)
# Define perturbation
ptb = PerturbationLpNorm(norm=np.inf, eps=0.1)
# Make the input a BoundedTensor with perturbation
my_input = BoundedTensor(my_input, ptb)
# Forward propagation using BoundedTensor
prediction = model(my_input)
```

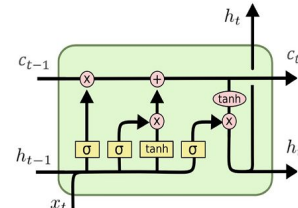
```
# Compute LiRPA bounds
lb, ub = model.compute_bounds(method="backward")
```

Any complex or
irregular network
can go here!

DenseNet, ResNeXt, WideResNet



Transformers



LSTMs

Compute bounds
directly!

auto_LiRPA: A Robustness Verification Library

 Colab Demo:

<http://PaperCode.cc/AutoLiRPA-Demo>

Demo includes an example of computing verification bounds for a **18-layer ResNet** model on **CIFAR-10** dataset. Once the ResNet model is defined as usual in Pytorch, obtaining provable output bounds is as easy as obtaining gradients through autodiff. Bounds are efficiently computed on GPUs.

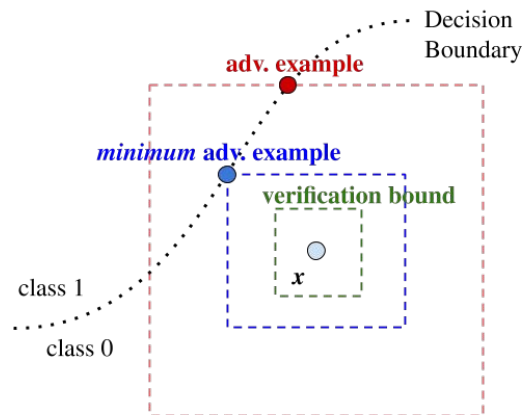


The auto_LiRPA library on GitHub:

<http://PaperCode.cc/AutoLiRPA>

Applications of CROWN: Certified Adv Defense

- Certified adversarial defense: using efficient verification techniques to enhance the robustness of a model, with provable robustness guarantees.
- At a high level, the green box is a function of network weights, so we can train the network to enlarge the box (more safe area)



Applications of CROWN: Certified Adv Defense

- Regular Training: empirical risk minimization

$$\min_{\theta} \mathbb{E}_{(x,y) \in \mathcal{X}} [\mathcal{L}_{\theta}(x, y)]$$

model parameter dataset Loss function (e.g., cross-entropy)

- Robust Training: robust minimax optimization

$$\min_{\theta} \mathbb{E}_{(x,y) \in \mathcal{X}} \left[\max_{x' \in \mathcal{S}(x)} \mathcal{L}_{\theta}(x', y) \right]$$

Worst case perturbation Perturbation set (e.g., ℓ_p norm ball around x) Adversarial example

Applications of CROWN: Certified Adv Defense

- How to solve robust minimax optimization?

$$\min_{\theta} \mathbb{E}_{(x,y) \in \mathcal{X}} \left[\max_{x' \in S(x)} \mathcal{L}_{\theta}(x', y) \right]$$

- **Adversarial training:** solve the inner max with gradient ascent; cannot guarantee to find the maximum due to non-convexity. No robustness guarantee.
- **Certified defense:** upper bound the inner max, and minimize the upper bound. If the upper bound is minimized, the original objective is guaranteed minimized.

Applications of auto_LiRPA: Certified Adv Defense

Certified defense with CROWN and **auto_LiRPA**:

$$\min_{\theta} \mathbb{E}_{(x,y) \in \mathcal{X}} \left[\overline{\mathcal{L}}_{\theta}(x, y) \right], \text{ where } \overline{\mathcal{L}}_{\theta}(x, y) \geq \max_{x' \in S(x)} \mathcal{L}_{\theta}(x', y)$$

↑
upper bound obtained by CROWN

- Treat the computation of $\mathcal{L}_{\theta}(x, y)$ as a computational graph, and upper bound its output given perturbed x
- $\mathcal{L}_{\theta}(x, y)$ can contain a complex neural network
- The upper bound can be **loose initially**, but can become quite **tight after training**

Certified Defense Results - MNIST

$\epsilon=0.3$, acc. under PGD attack=93.95%, verified acc.=92.98%



$\epsilon=0.4$, acc. under PGD attack=90.53%, verified acc.=87.94%



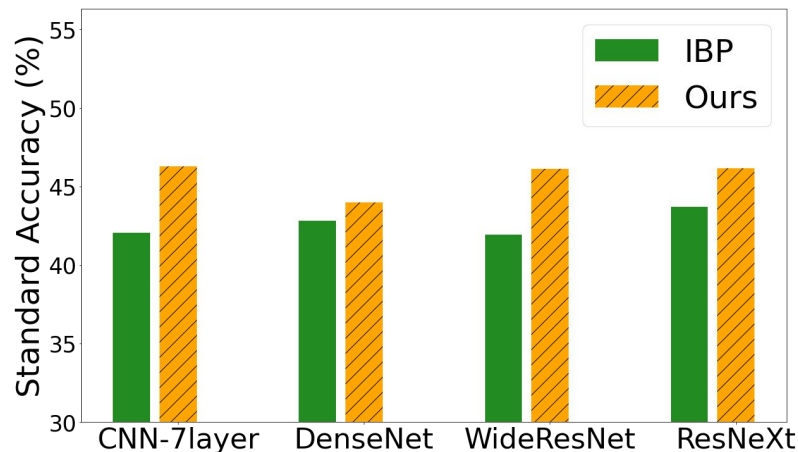
Input image range:
[0, 1]

Verified acc. is an **guaranteed lower bound** of acc. under **any** attack

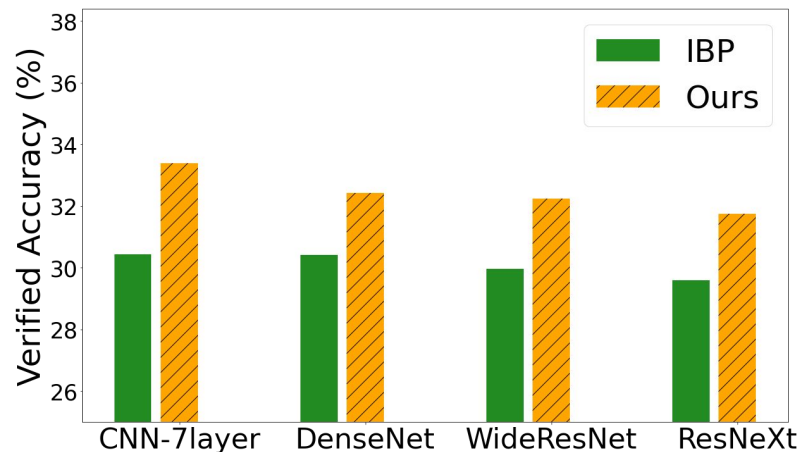
Certified Defense Results - CIFAR-10

CIFAR-10 results: consistent improvements on both standard accuracy and verified accuracy thanks to tighter bounds

Standard accuracy (higher is better)



Verified accuracy (higher is better)



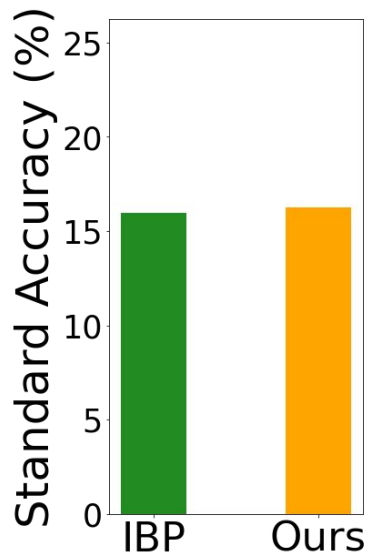
IBP = "interval bound propagation" (Gowal et al. 2019)

ℓ_∞ norm perturbation $\epsilon = \frac{8}{255}$

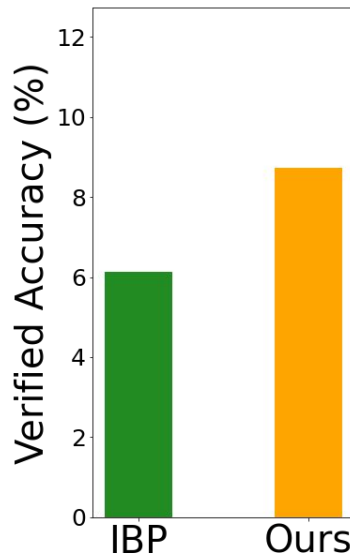
Certified Defense Results - ImageNet

ImageNet (64*64) results: better than IBP, but still a very challenging problem.

ImageNet: standard accuracy



ImageNet: verified accuracy



Verified accuracy is low (<10%) but this is a very challenging setting since we have 1000 class labels

WideResNet

l_∞ norm perturbation

$$\epsilon = \frac{1}{255}$$

Challenges in Certified Defense

- Certified defense typically has **much worse clean accuracy** than normally or adversarially trained models
- Bound propagation based certified defense is **difficult to scale** to modern large networks (e.g., ResNet-50)
- **Verified accuracy is low** (except for MNIST), still has a large room for improvement
- Can we train our model with adversarial training to get better clean accuracy, and verify it using CROWN?

Back to the Verification Problem

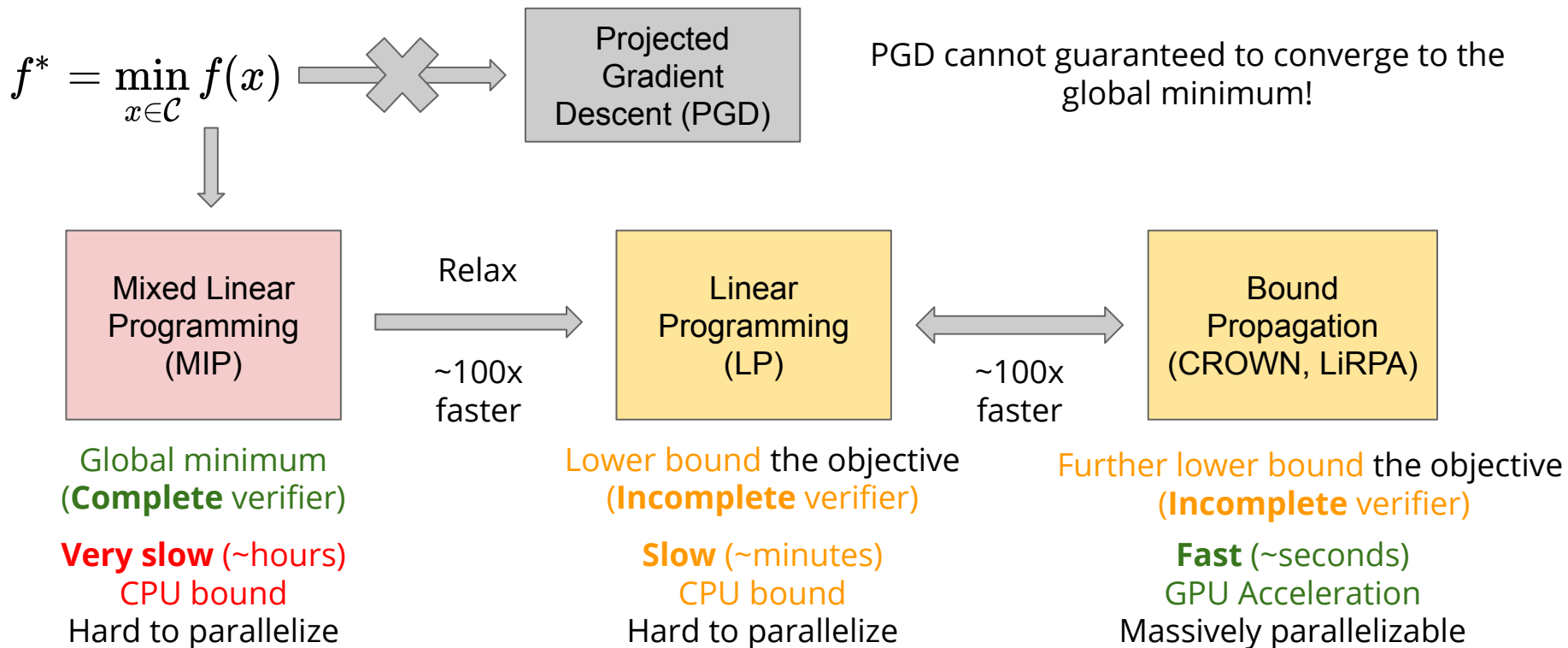
Assuming $f(x_0) > 0$, we solve

$$f^* = \min_{x \in \mathcal{C}} f(x)$$

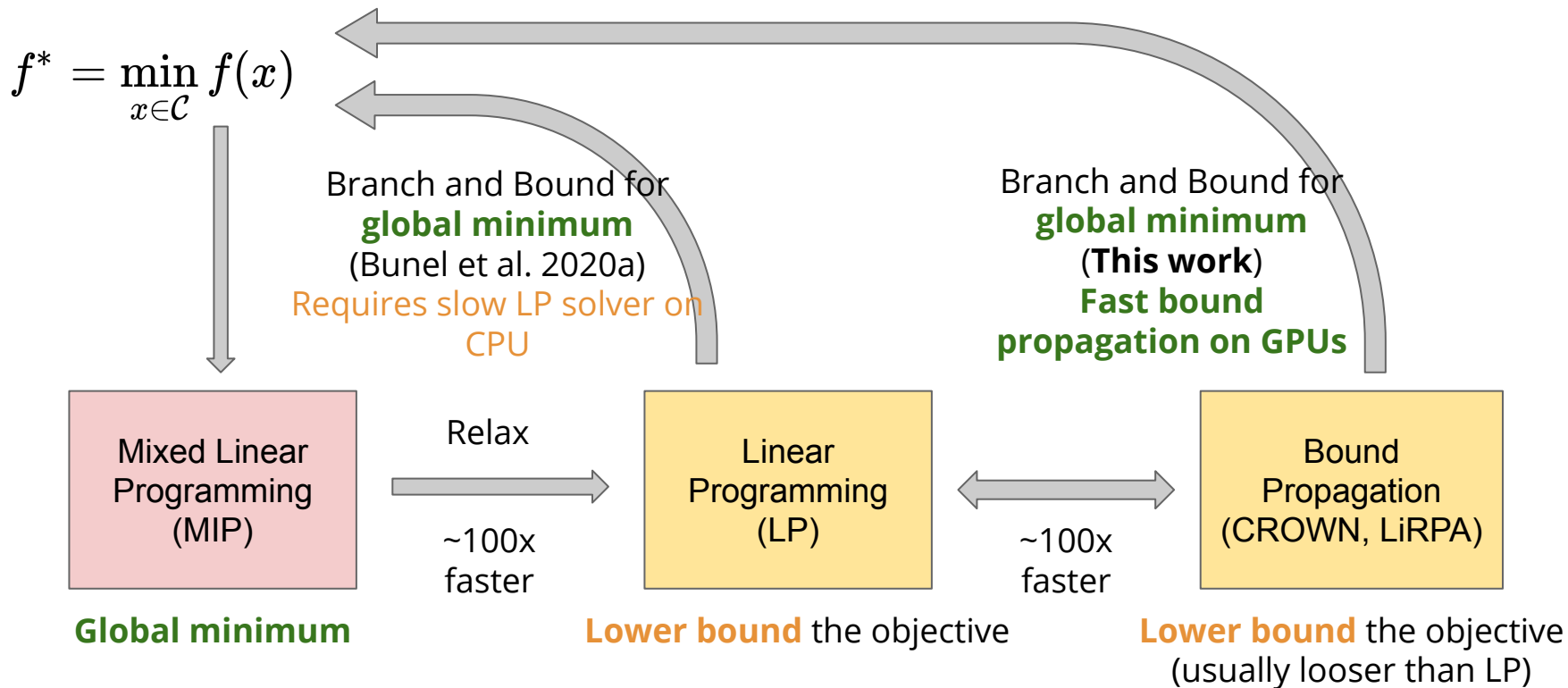
\mathcal{C} is usually a set “around” x_0 , e.g., $\mathcal{C} := \{x \mid \|x - x_0\|_p \leq \epsilon\}$

- Incomplete verifiers are **usually too weak** to give tight bounds on models trained via adversarial training.
- **Complete** verification: solve exactly (typically NP-complete)
- **Incomplete** (or relaxed) verification: find a *lower bound* of f^*

From Incomplete to Complete Verification



From Incomplete to Complete Verification



From Incomplete to Complete Verification

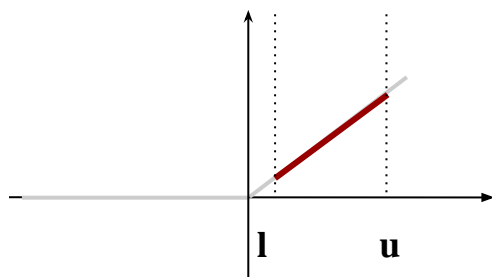
Our approach:

Combine **rapid bound propagation** based incomplete verifiers (CROWN) on GPUs with **branch and bound (BaB)** to achieve the same power of complete verifiers and **scale up** neural network verification

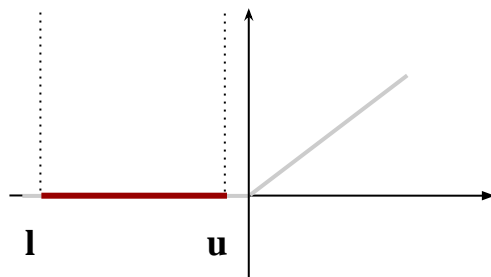
Branch and Bound for ReLU Network Verification

Goal: prove $f(x) > 0, \forall x \in \mathcal{C}$

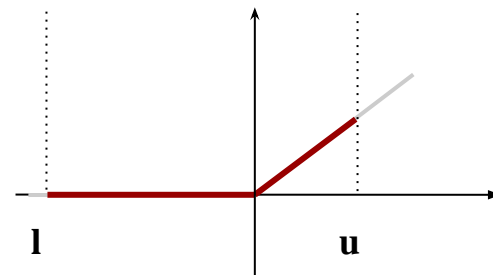
ReLU neurons have three cases depends on pre-activation bounds:



Always active (linear)



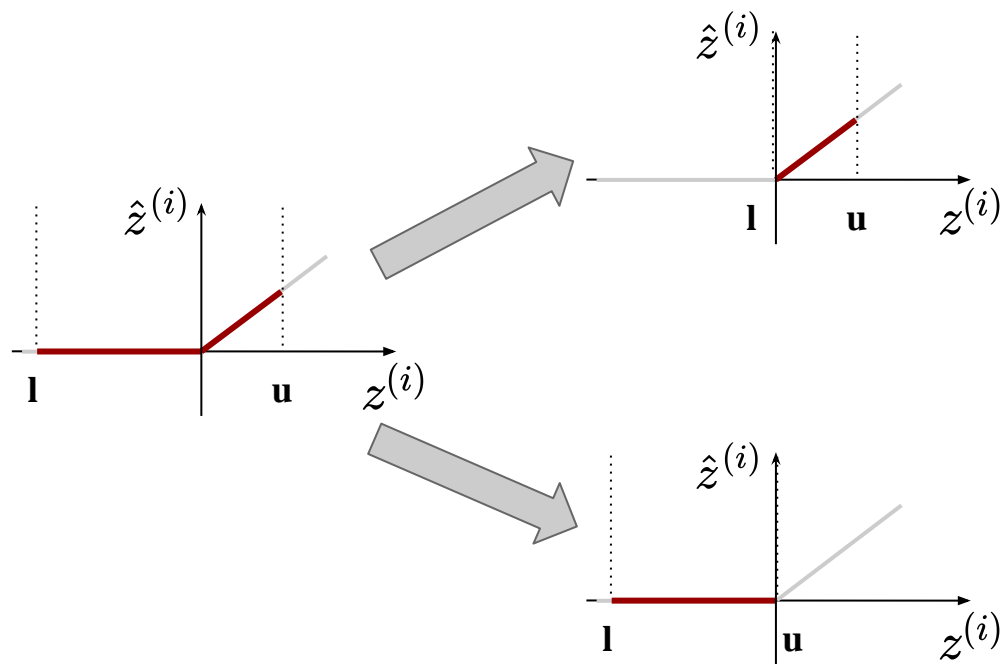
Always inactive (zero)



Unstable (non-linear)
Must be relaxed, e.g., using
linear lower and upper bounds

Branch and Bound for ReLU Network Verification

Split each “unstable” ReLU neurons to two cases:



Additional linear constraint
 (“split constraint”):

$$z^{(i)} \geq 0$$

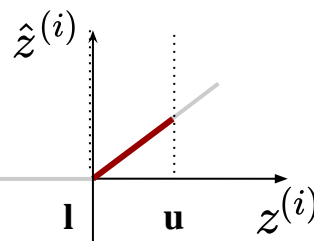
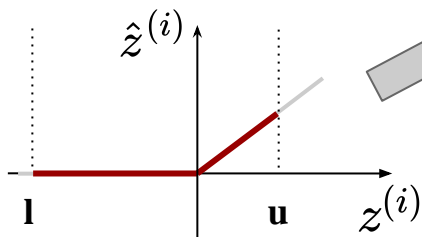
OR

$$z^{(i)} < 0$$

Branch and Bound for NN Verification

Using an incomplete solver (e.g., CROWN) to check each case:

$$f_{\text{CROWN}}^* = -0.5$$

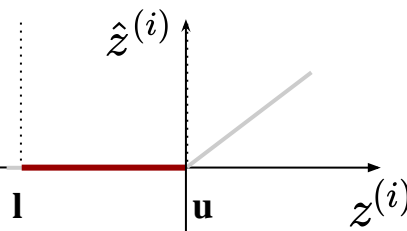


$$z^{(i)} \geq 0 \quad f_{\text{CROWN}}^* = -0.1 < 0$$

split constraint

Split next unstable
ReLU neuron...
(build a search tree!)

Goal: prove $f^* > 0$



$$z^{(i)} < 0 \quad f_{\text{CROWN}}^* = 0.1 > 0$$

Verified



Bound Propagation for Complete Verification

Pros:

- Much faster than LP based verifier (~100x)
- Massively parallelizable and GPU friendly

Cons:

- Bounds are usually looser than typical LP based verifiers
- Cannot directly handle split constraints, leading to infeasible splits and incompleteness

$$z^{(i)} \geq 0$$

$$z^{(i)} < 0$$

Bound Propagation for Complete Verification

Bounds are usually looser than typical LP based verifiers

Solution: tighten bounds using gradients (obtained by autodiff)

$$f^* = \min_{x \in \mathcal{C}} f(x) \geq \max_{0 \leq \alpha \leq 1} \min_{x \in \mathcal{C}} f_{\text{CROWN}}(x; \alpha)$$

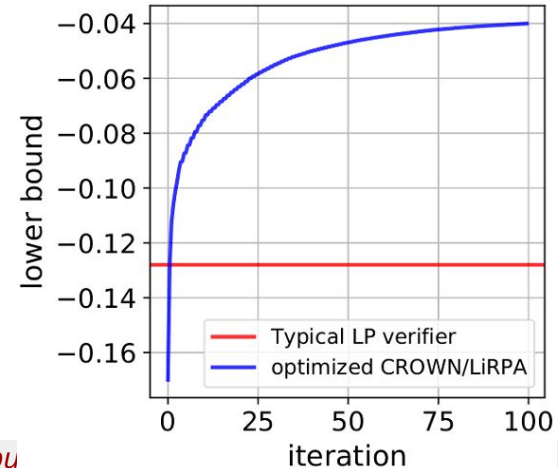
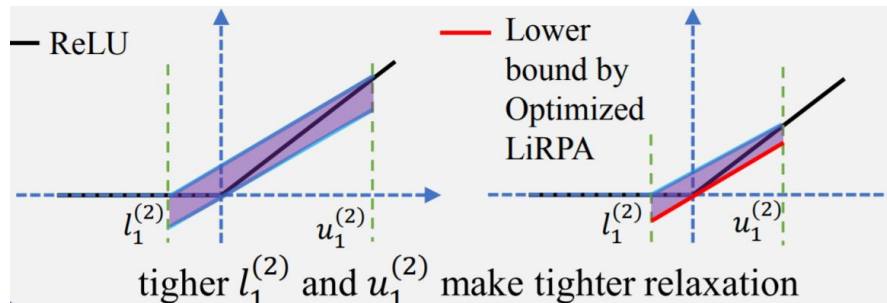
Inner minimization can be solved in closed form

- Salman et al. 2019 showed that free parameters α in CROWN correspond to dual variables in LP
- Optimizing α leads to a tighter bound
- We can optimize **both intermediate layer bounds** and the final lower bound using a single objective

Bound Propagation for Complete Verification

Bounds can be tightened, sometimes better than typical LP verifiers

- Typical LP based verifiers cannot cheaply optimize intermediate layer bounds
- We can tighten the relaxation during the optimization process



Bound Propagation for Complete Verification

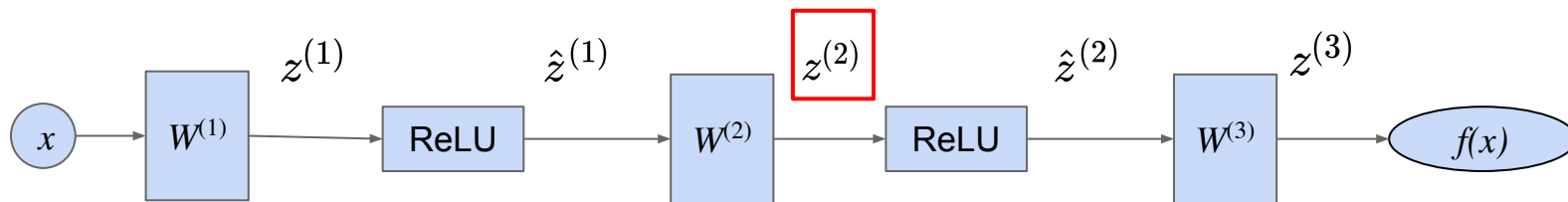
- **β -CROWN** can handle the **split constraints** in branch and bound (BaB), allowing combining bound propagation based method with BaB for **very efficient GPU** based complete verification
- **β -CROWN** also allows **optimization of bounds** (e.g., update certain parameters in bound propagation, to achieve tighter bounds), to potentially achieve **tighter bounds than LP**

Beta-CROWN: *Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Verification.* **Shiqi Wang***, **Huan Zhang***, **Kaidi Xu***, Xue Lin, Suman Jana, Cho-Jui Hsieh, J. Zico Kolter (*Equal contribution). <https://arxiv.org/pdf/2103.06624.pdf>

Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers, Kaidi Xu*, Huan Zhang*, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, Cho-Jui Hsieh. ICLR 2021.

β -CROWN Bound Propagation

- We always maintain a linear lower bound for **output neuron** w.r.t. **hidden or input neuron**.



CROWN:
$$\min_{x \in \mathcal{C}} f(x) \geq \min_{x \in \mathcal{C}} W^{(3)} D^{(2)} z^{(2)} + \text{const.}$$

β -CROWN:
$$\min_{x \in \mathcal{C}, z^{(2)} \in \mathcal{Z}} f(x) \geq \max_{\beta \geq 0} \min_{x \in \mathcal{C}} W^{(3)} D^{(2)} z^{(2)} + \beta^\top S^{(2)} z^{(2)} + \text{const.}$$

Lagrangian/KKT multipliers

β -CROWN Bound

β -CROWN main theorem:

$$\min_{x \in \mathcal{C}, z \in \mathcal{Z}} f(x) \geq \max_{\beta \geq 0} \min_{x \in \mathcal{C}} (\mathbf{a} + \mathbf{P}\beta)^\top x + \mathbf{q}^\top \beta + c,$$

Inner minimization can be solved in closed form (q is dual norm):

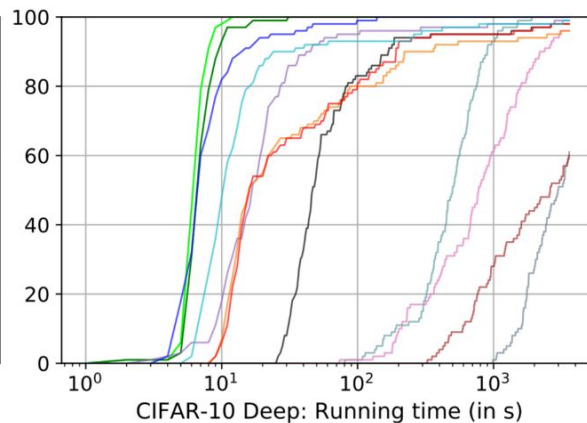
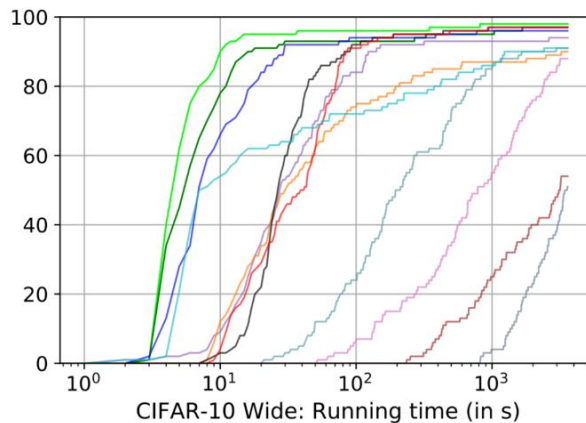
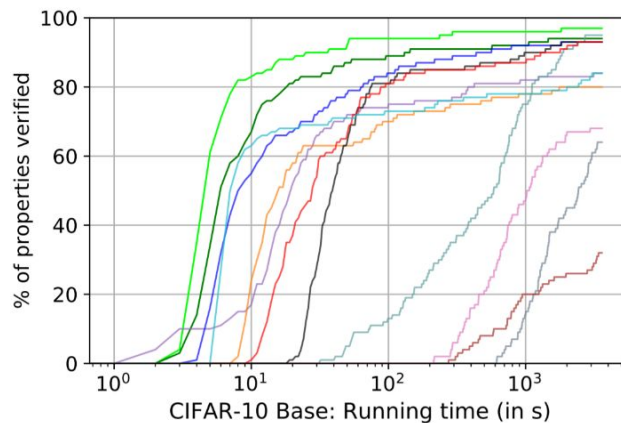
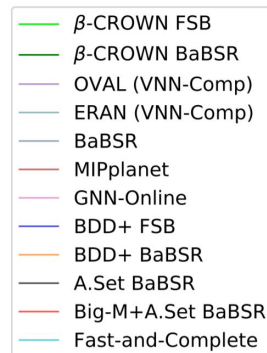
$$\min_{\substack{x \in \mathcal{C} \\ z \in \mathcal{Z}}} f(x) \geq \max_{\beta \geq 0} - \|\mathbf{a} + \mathbf{P}\beta\|_q \epsilon + (\mathbf{P}^\top x_0 + \mathbf{q})^\top \beta + \mathbf{a}^\top x_0 + c := \max_{\beta \geq 0} g(\beta)$$

Also, recall that CROWN has an optimizable parameter α

$$\min_{x \in \mathcal{C}, z \in \mathcal{Z}} f(x) \geq \max_{0 \leq \alpha \leq 1, \beta \geq 0} g(\alpha, \beta) \quad \alpha, \beta \text{ optimized using gradient ascent}$$

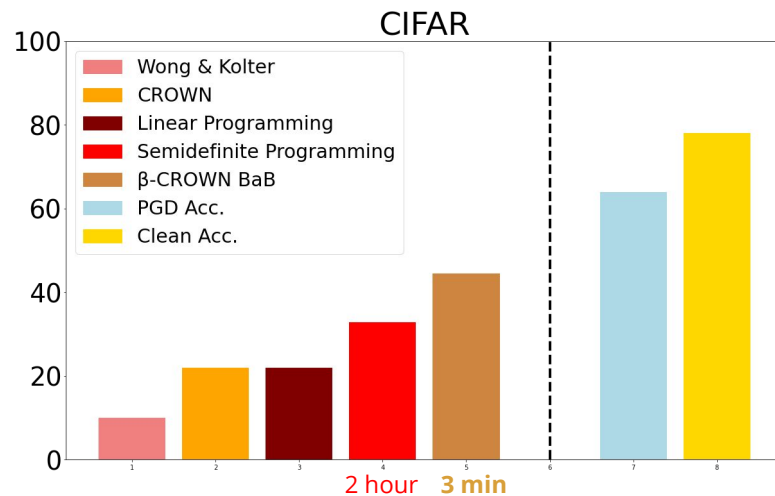
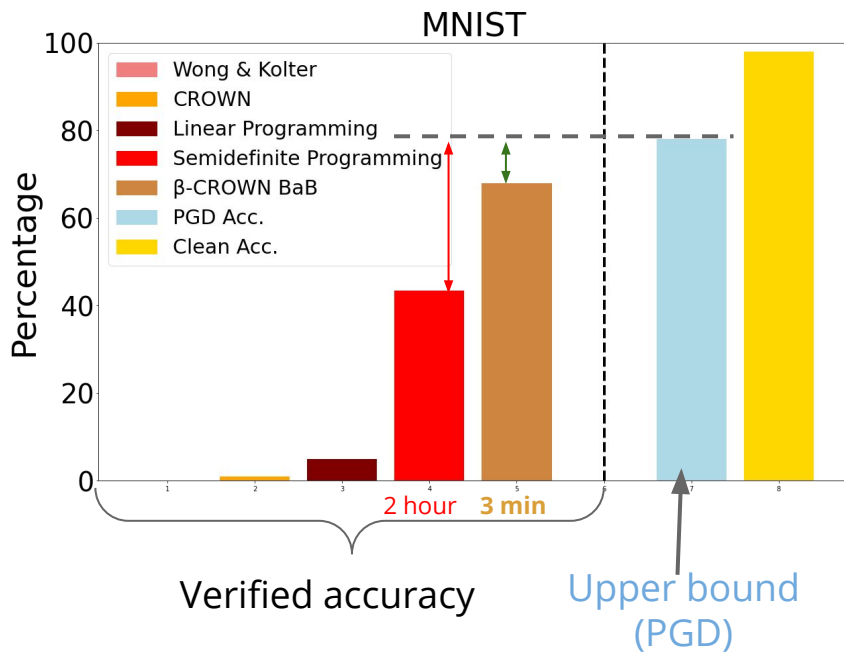
β -CROWN: Complete Verification Benchmark

- CIFAR-10 benchmark with 300 examples to verify (VNN COMP competition 2020)
- β -CROWN with branch and bound is **2 to 3 magnitudes faster** than classic LP based CPU verifiers (e.g., MIPplanet, Ehlers 2017)



β -CROWN on Adversarially Trained Models

- β -CROWN with BaB achieves the best verified accuracy, closing the gap between verified accuracy and PGD attack accuracy



Adversarially trained models are from (Dathathri et al. 2020)



Thank you!

PyTorch Code:

[PaperCode.cc/AutoLiRPA](https://papercode.cc/AutoLiRPA) (robustness verification library)

[PaperCode.cc/BetaCROWN](https://papercode.cc/BetaCROWN) (complete verification)



期刊论文征稿 (Deadline: 07/15/2021)

Frontiers in Big Data - Trustworthy Machine Learning

see [PaperCode.cc/Frontiers](https://papercode.cc/Frontiers) for details

References

Athalye, Anish, Nicholas Carlini, and David Wagner. "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples." *International Conference on Machine Learning*. PMLR, 2018.

Ehlers, Ruediger. "Formal verification of piece-wise linear feed-forward neural networks." *International Symposium on Automated Technology for Verification and Analysis*. Springer, Cham, 2017.

Wong, Eric, and Zico Kolter. "Provable defenses against adversarial examples via the convex outer adversarial polytope." *International Conference on Machine Learning*. PMLR, 2018.

Salman, Hadi, et al. "A convex relaxation barrier to tight robustness verification of neural networks." *arXiv preprint arXiv:1902.08722* (2019).

Zhang, Huan, et al. "Efficient Neural Network Robustness Certification with General Activation Functions." *Neural Information Processing Systems (NeurIPS)* (2018).

Singh, Gagandeep, et al. "An abstract domain for certifying neural networks." *Proceedings of the ACM on Programming Languages* 3.POPL (2019): 1-30.

Bunel, Rudy, et al. "A unified view of piecewise linear neural network verification." *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018.

Bunel, Rudy, et al. "Lagrangian decomposition for neural network verification." *Conference on Uncertainty in Artificial Intelligence*. PMLR, 2020.

References (cont.)

Zhang, Huan, et al. "Towards stable and efficient training of verifiably robust neural networks." *ICLR 2020*.

Dathathri, Sumanth, et al. "Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming." *Neural Information Processing Systems (NeurIPS)* (2020).

Lu, J., and P. Mudigonda. "Nueral network branching for nueral network verification." *Proceedings of the International Conference on Learning Representations (ICLR 2020)*. Open Review, 2020.

Xu, Kaidi, et al. "Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers." *arXiv preprint arXiv:2011.13824* (2020).

De Palma, Alessandro, et al. "Scaling the Convex Barrier with Active Sets." *arXiv preprint arXiv:2101.05844* (2021).

Wang, Shiqi, et al. "Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification." *arXiv preprint arXiv:2103.06624* (2021).